# METHOD TO SECURE AN ELECTRONIC ASSEMBLY AGAINST ATTACKS BY ERROR INTRODUCTION

This invention concerns a method to secure an electronic assembly implementing any algorithm where a check must be carried out to ensure that the algorithm was executed correctly, whether for the intermediate steps or for the intermediate data. More precisely, the purpose of the method is to produce a version of the algorithm which is not vulnerable to certain types of attack through introduction of one or more errors - known as *Differential Fault Analysis* or *Extended Fault Analysis* - which attempt to obtain information about one or more data items or operations involved in the algorithm calculation by studying the calculation procedure of the electronic assembly when one or more errors are introduced.

## TECHNICAL FIELD

The algorithms considered in the remainder of the document are given as non-limiting examples of cryptographic algorithms using a secret key to calculate output information according to input information; it may concern an encryption, decryption, signature, signature check, authentication or non-repudiation operation. They are built so that an attacker, knowing the inputs and the outputs, is unable in practice to deduce any information concerning the secret key itself.

However, this invention covers those algorithms where tests must be carried out to ensure that all intermediate steps and data involved in the calculation are free from errors.

The first time attacks using error introduction appeared dates back to 1996:

- Ref (0): New Threat Model Breaks Crypto Codes – D. Boneh, R. DeMillo, R. Lipton – Bellcore Press Release

An example of this type of attack on the cryptographic algorithm with DES secret key -ref (1)- can be found in article -ref (2)- published by Eli Biham and Adi Shamir in 1996.

- Ref (1): FIPS PUB 46-2, *Data Encryption Standard*, 1994
- Ref (2): A New Cryptanalytic Attack on DES, Draft

The invention proposed also concerns extended attacks such as those where the attacker uses the fact that certain functions have an output which is smaller than the input; there are several inputs which give the same output. Consequently, by modifying the input, a correct result can still be obtained at the output, which is sometimes interesting.

The purpose of the method according to this invention is to eliminate the risks of DFA attacks on electronic assemblies or systems by modifying the functions involved in the calculation.

Another purpose of this invention is to modify a cryptographic calculation process implemented by cryptography electronic assemblies protected so that the above-mentioned basic assumption is no longer satisfied, i.e. no intermediate variable or function where an error could be introduced remains undetected by the system.

## SUMMARY OF THE INVENTION

This invention concerns a method to secure the execution of a program in an electronic assembly comprising information processing means and information storage means, characterised in that it consists in performing an additional calculation by a verification function on at least one intermediate result in order to obtain a calculation signature.

This invention also concerns an electronic assembly and for example a smart card and a program used to implement said method.

## BRIEF DESCRIPTION OF THE DRAWINGS

Other purposes, features and advantages of the invention will appear on reading the description which follows of the implementation of the method according to the invention and of a mode of realisation of an electronic assembly designed for this implementation, given as a non-limiting example, and referring to the attached drawings in which:

- figures 1 and 2 are a diagrammatic representation of a mode of realisation of the method according to this invention.

## WAY OF REALISING THE INVENTION

The purpose of the method according to the invention is to secure an electronic assembly and for example an onboard system such as a smart card implementing a cryptographic calculation process using a secret key. The electronic assembly includes information processing means such as a processor and information storage means such as a memory. The cryptographic calculation process is installed in the memory, for example ROM type, of said assembly. The processor of said system executes the calculation process by using a secret key, stored in a secret area of a memory, EEPROM type for example.

The method to secure said electronic assembly implementing a traditional calculation process which must be error free, subject of this invention and illustrated on figures 1 and 2, is remarkable in that the functions f implemented in the calculation are modified by more general functions f' (known as "super-functions") but where it is easy to find the normal result of the function f. Any error introduced in the calculation will be detected by a verification function V associated with the "super-function". These super-functions f' and verification functions V will obviously be described below and numerous examples will also be given.

The principle of super-function f' according to this invention is described below, referring to figures 1 and 2. An algorithm executed by an electronic device is always a series of elementary operations. We will describe here the principle of super-function on one of these elementary

operations. Any elementary operation can be described as a function *f* of a finite set E to a finite set F (figure 2). The principle of the super-function is to consider a super-function f' of E' to F' where:

- E' is a set such that there exist a one-to-one mapping $h_1$ of E in E' (i.e. if we take two different elements in E, their images by $h_1$ are also different in E'). In practice, this is the same as saying that E' has at least as many elements as E.
- F' is a set such that there exist an onto mapping $h_2$ of F' in F (i.e. for all elements *y* of F there exist *x* elements of F' such that $h_2(x)=y$). In practice, this is the same as saying that F' has at least as many elements as F.
- And in particular, for any element x of E, the following equality must be true: $h_2(f'(h_1(x))) = f(x)$. This is the same as saying that the function f can be calculated by using the calculation of the function f' and by the super-sets E' and F'.

Error detection is carried out by a verification function V. A verification function is one used to check the intermediate results (or those considered to be critical) of an algorithm.

Formally, if we write:

- $\boldsymbol{B} = \{0,1\}^*$ the set of binary words.
- $E = \{x_i\}_{0<i<n+1}$ where the elements $x_i$ are in practice elements of ***B,*** the set of n intermediate results that we want to protect.

A verification function V is a function which associates a binary word of fixed length N with every element of E: $V: E \rightarrow \{0,1\}^N$

In practice, in more concrete terms:

- V is a function which associates a calculation "signature" with a set of intermediate values; by repeating this calculation, this signature will be used to detect a possible error.
- We can clearly see that in practice N will be of the order of 8, 32 or 64 (usual size in computing) and that since the set of values to be protected is larger, the verification function cannot be one-to-one. Consequently, different calculations could give similar results by V (this property will be called

collision in the remainder of the document). V must therefore be chosen so that the collisions are as infrequent as possible and well distributed.

The security principle according to this invention consists in combining the two methods previously described: firstly the principle of super-function is applied, in order to protect the functions individually, then a verification function. This produces a calculation signature. All or part of the calculation can now be repeated one or more times in order to recalculate a new signature, then the signatures are compared in order to detect any errors. The verification function performs a calculation on at least one intermediate result from an operation of the calculation process or of a super-function. An intermediate result is a result obtained during execution of a cryptographic calculation process, as opposed to the final result. As an illustration, according to a special form of realisation, it is recommended to apply one or more verification functions on intermediate results obtained at sensitive points during execution of the algorithm considered.

A first example of applying the security method according to this invention to the DES algorithm is given below.

Concerning the super-functions, only one is used in order to protect the S-boxes. The S-boxes have indeed one 6-bit input and one 4-bit output. Several 6-bit values may therefore give the same result on 4 bits, which an attacker can use. In the present case:

- $E = \{0.1\}^6$ and $F = \{0.1\}^4$

  We then introduce
- $E' = E$ and $h_1$ identity function
- $F' = \{0,1\}^6$ and $h_2$ which simply consists of removing the most significant bit and the least significant bit of a word of F.
- f' is built as follows: if x is an element of E' (therefore on 6 bits), the first bit and the last bit of f'(x) are those of x; the four bits in the middle are those given by the usual result of the boxes. A brief analysis of the operating mode used with the S boxes shows that for all elements of E we have: $h_2(f'(h_1(x)) = \text{S-box}(x)$. Concerning the verification function, it is first used on the results output from each round (64 bits) and on the

concatenated outputs of the function f' (48 bits which will be considered as 64).

The verification function as such must then be chosen: several examples are given below.

We write in the remainder of the document $X = \{x_i\}_{0<i<n+1}$ the set of values to be checked and V the verification function. The intermediate verification value taking into account the first j values $x_i$ is sometimes written $V_j$. We therefore have $V(x) = V_n$

A first example of verification function is the concatenation of values: this means that a trace is kept throughout the calculation, which is costly in terms of memory but which is efficient since there is no risk of collision.

$$V_{j+1} = V_j \,||\, x_{j+1}$$

A second example is the exclusive or of the values:

$$V(x_1, \ldots, x_n) = x_1 \text{ XOR } \ldots \text{ XOR } x_n$$

A third example is addition modulo $2^N$ (N is the length of the result). This is the same as adding the intermediate results and truncating the result at the required length:

$$V_{j+1} = ( V_j + x_{j+1} ) \text{ modulo } 2^N$$

A fourth example is the usual CRC of the values:

$$V(x_1, \ldots, x_n) = \text{CRC } (x_1, \ldots, x_n)$$

A fifth example is the exclusive or of the values where a cyclic shift is carried out on the intermediate result of a random number of bits (e.g. 1):

$$V_{j+1} = (V_j << 1) \text{ XOR } x_{j+1}$$

A sixth example is the exclusive or of the values with multiplication. At each step, an XOR is carried out on the intermediate verification value with the product of the truncated intermediate verification value and the current $x_i$.

$$V_{j+1} = V_j \text{ XOR } ((V_j \times x_{j+1}) \text{ modulo } 2^N)$$

A seventh example is data hashing by a cryptographic function. For example, all the values can be concatenated, hashed using the SHA-1 function and the required number of bits of the result can be kept.

V = SHA-1(X)

A second example of applying the security method according to this invention to the RSA algorithm is given below, in which the following loop, using variables M,C1,C2,N, and a bit string D, is sometimes executed.

```
C1 = 1
For i = 0 to N do:
        C2 = C1 x C1
        C1 = C2 x M
        If bit i of D = 0 then
          C1 = C2
        End If.
End loop.
Output the result C1.
```

We can see that if the attacker modifies for example C1 before the "If" and if the If instruction is executed, the result is correct: the attacker can therefore obtain information concerning the value of D.

No super-functions are used in this case; application of a simple exclusive or on the intermediate values C1 and C2 will detect any errors.

Sometimes, the attacker does not control the error introduction fully. By working on a larger space and by checking the consistency of the results, an error introduced can therefore be detected since it generates an impossible result. For example, if two eight-bit numbers are added and the result is stored on 16 bits, an error introduced on the result has a strong change of affecting the 7 most significant bits (generally 0 using the laws of arithmetic) which means that the error will be detected.